Quick Union (II)

Domingo Gómez Pérez

Problema con quick-find

El problema con quick-find es al hacer la operación unir, hay que recorrer todo el array y hacer varios cambios.

Problema con quick-find

El problema con quick-find es al hacer la operación unir, hay que recorrer todo el array y hacer varios cambios.

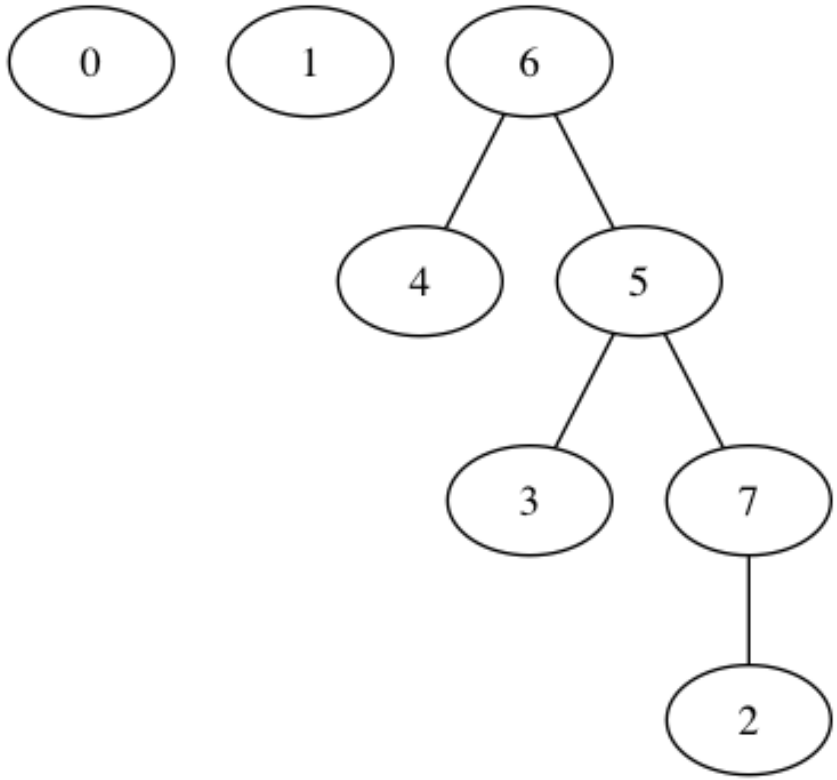
¿Qué podemos hacer para añadir un nodo a una componente?

Quick-union (primera aproximación)

Utilizamos un array ID:

- Cada componente conexa va a ser representada por un árbol.
- La componente conexa esta representada por la raíz del árbol.
- $ID[p]$ contiene al padre en el árbol, a menos que sea la raíz, en ese caso es el mismo.

Quick-union (primera aproximación)




```
ID = [0,1,6,6,6,6,6];
```

Quick-union (primera aproximación)

- Unir: solamente tenemos que cambiar el padre de uno de los nodos a ponerlo como hijo del otro padre.
- Conectados: solamente tenemos que buscar el padre de los dos nodos y compararlos.

Quick-union (código)

```
encontrar_raiz(n):
mientras id[n] != n:
  n = id[n]
return n

Conectados(p,q):
return encontrar_raiz(p) == encontrar_raiz(q)

unir(p,q):
pid = encontrar_raiz(p)
qid = encontrar_raiz(q)
id[pid] = qid
```

Quick-union (problemas)

- Conectados es demasiado lento.
- unir es demasiado lento.

Siguiente iteración

El problema es que los árboles se pueden volver muy altos.

¿Cómo podemos hacer que los árboles crezcan menos?

Pregunta

Dada las siguientes ordenes,

```
unir(0,1); unir(2,0);unir(7,2);unir(3,7);unir(4,3);
```

¿cómo debería hacerse el árbol para que quede de la "mejor" forma posible? ¿y la "peor"?

Siguiente iteración

- añadir los nodos a árboles y guardar el número de nodos en el árbol en un array adicional,
- tener en cuenta el número de nodos al cambiar las raíces,
- solo hay que cambiar el número de nodos de uno de los árboles.

Siguiente iteración (código)

```
encontrar_raiz(n):
mientras id[n] != n:
  n = id[n]
return n

Conectados(p,q):
return encontrar_raiz(p) == encontrar_raiz(q)

unir(p,q):
pid = encontrar_raiz(p)
qid = encontrar_raiz(q)
if pid != quid:
  if tamaño[pid] <tamaño[qid]:
     id[pid] = qid
  else:
     id[qid] = pid
```

Complejidad

- Conectados: depende de la altura de los árboles
- Unir: es constante, dadas las raíces

Teorema

Dados N objetos, y después de un número finito de inserciones, la altura de cualquiera de los árboles que se han generado es del orden logarítmico.

Prueba

La idea de la prueba es que los árboles solo crecen cuando son añadidos a otro árbol con más nodos.

Prueba

La idea de la prueba es que los árboles solo crecen cuando son añadidos a otro árbol con más nodos. ¿Cuantos nodos tendrá, como mínimo, el nuevo árbol? ¿cuantas veces se puede repetir este proceso?

Podríamos parar pero. . . .

Aquí podríamos parar pero, se puede mejorar:

- La idea es que cuando recorramos un árbol poner el padre de cada nodo a abuelo.
- Eso hace el árbol más plano.
- El análisis de este algoritmo es interesante, empezando en N objetos y realizando M uniones, hay que realizar del orden de

$$N + M \log^* N$$

.

Percolación

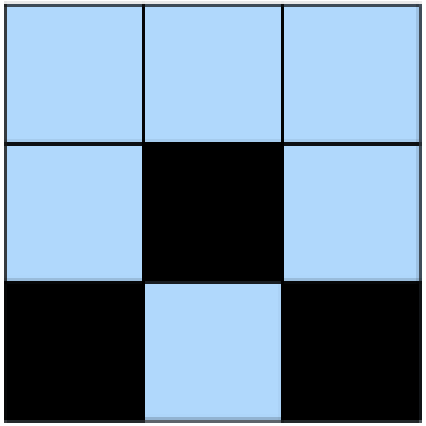
En este problema, hay una transición de fase (es lo que se deduce del los experimentos), para N = 20:

- Cuando p es menor que 0.593, casi siempre percola.
- Cuando p es mayor que 0.593, casi nunca percola.

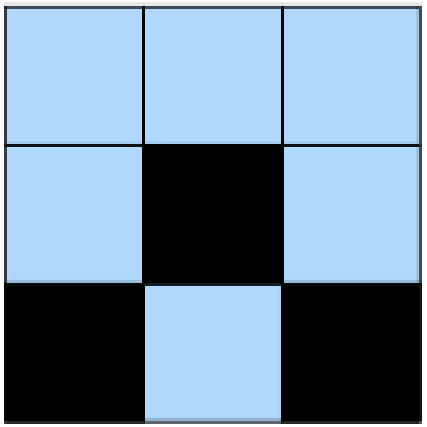
Simulación utilizando métodos Monte Carlo

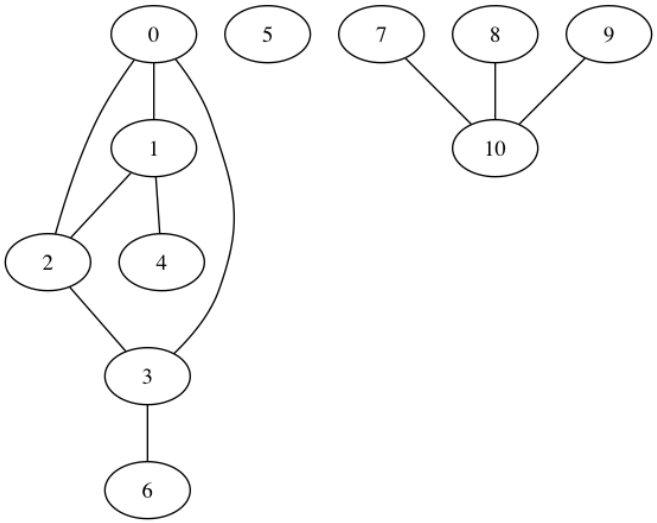
- Empezar con una matriz totalmente llena.
- seleccionar a cada paso un casilla llena y liberadla.
- Si el sistema percola, contar el porcentaje de casillas libres y parar.
- En otro caso, repetir.

Algoritmo



Algoritmo





Pregunta

- Suponed que en el ejemplo anterior, se cambia la casilla 5 ¿que habría que hacer?
- ¿Con que linea de código se puede comprobar que el sistema percola?